John von Neumann Institute for Computing

Performance comparison and optimization: Case studies using BenchIT

R. Schöne, G. Juckeland, W.E. Nagel, S. Pflüger, R. Wloch

published in

Parallel Computing:
Current & Future Issues of High-End Computing,
Proceedings of the International Conference ParCo 2005,
G.R. Joubert, W.E. Nagel, F.J. Peters, O. Plata, P. Tirado, E. Zapata (Editors),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **33**, ISBN 3-00-017352-8, pp. 877-884, 2006.

http://www.fz-juelich.de/nic-series/volume33

Performance comparison and optimization: Case studies using BenchIT

R. Schöne[a], G. Juckeland[a], W. E. Nagel[a], S. Pflüger[a], R. Wloch[a]

[a]Center for Information Services and High Performance Computing, Technische Universität Dresden, 01062 Dresden, Germany

Quite often, efficient usage of computing resources is a challenge. Performance measurements, comparisons, and the resulting optimizations are a suitable way to accomplish this goal. The BenchIT project provides a framework for performance measurements on UNIX based systems. Our approach combines low requirements on software with a simple interface for measuring kernels and a strict separation of configuration, compiling, measuring, and result evaluation. Unlike most other benchmark systems, the BenchIT environment provides functions on several levels: Measurements on varying problem sizes, graphical presentation of the results, and the capability to automatically compare with other measurements. Those characteristics along with the userfriendly interfaces which allow access to the database, make comparing different algorithms easy when using BenchIT. This paper presents the BenchIT platform and describes first results on selected machines.

Introduction

Performance measurement is complex and frequently discussed in the analysis of computer systems. The diversity and popularity of benchmarks are now part of our ”computer culture”. But how can different systems be compared to each other? Sure, there are standard benchmarks like LINPACK and SPEC - general and built for many systems. However, they have shortcomings. Namely, just the single resulting number, which is in SPEC referred to as the performance of a basic system, and in LINPACK to the achieved GFLOPS. The question is if one single number is enough to specify and categorize a whole computer system. On one hand, having only one value gives the user an abstract overview about the general performance in specific areas. But on the other hand, perhaps one number is not enough. Every user has different requirements regarding a computer system, and the predominance in one specific problem class could be important to him. BenchIT is designed to provide an abstract interface for comparing computer systems. It enables the user to benchmark nearly every possible algorithm on UNIX systems, providing an infrastructure to analyze the results cooperatively. One special test case is the multiplication of dense double precision floating-point matrices which is the subject of further performance considerations in section 3.
Opteron and Itanium based systems form 20 percent of the processors in the actual 25th TOP500 list, as well as 20 percent of the total achieved performance. In this paper we show firstly the influence different compiler flags have on the performance of one single processor and secondly the influence they have on the performance of different BLAS libraries and OpenMP.

1. Measurements with BenchIT

As previously mentioned in other publications ([1], [2]), BenchIT identifies systems with ”LOCALDEF” files, which are written in plain text. They contain a lot of information about the system and are separated into three files. One file holds compiler-and runtime-options like compiler names, flags, libraries, maximum runtime, etc. The next file contains information about the systems architecture which is needed to compare different machines, for instance processor name, clock rate,

memory type.
A tool for semi-automatic generation of this file called ”Architecture Information Database” (A.I.D), is under construction and will be available soon. The last file contains information about how the results are visualized when using gnuplot. These files can be extended and customized in such a way that additional information about the system is added to the database automatically. When trying to compile measurement kernels without existing LOCALDEFS, a routine will try to auto-detect or predefine those settings (See figure 1). After the choosing basic settings like the C-compiler and the processor name, the measurement kernels can be started. The measurement kernels are compiled using a central script, which writes the created executables to a separate folder for binaries. While being compiled, the kernels store the systems environment variables inside the binary. When being executed by the runscript, they restore the original environment from the compile time. The advantages are obvious: Having a strict separation of compilation and execution, it is possible to start the processes on batch systems with exactly the same environment as when the kernel was compiled. It simply works better with cross-compilers and compiled versions still available for measurements later.
The compiled kernels can be measured on several code compatible systems by resetting the differing variables. Furthermore, the precision of measurements can be increased by changing the number of measurement cycles and by using performance counter libraries like PAPI [3] or PCL [4]. Skeletons for measurement-kernels are also available for use of these libraries.
When measurements finished, results thus obtained can be used to create different graphic files locally or to upload them to the website[1], where they can be compared with other peoples results. It is possible to share them with other people and groups.
Each step can also be done with a GUI, which helps novice users but also accelerates the work of professionals. It helps filling out the LOCALDEFS with a graphical editor, allows editing and building new measurement kernels with a build in IDE[2], starting them by providing a graphical interface for the kernels and plotting the results.
With the GUI, it is possible to run jobs on other systems using standard tools like ssh and tar. It is also possible to run the GUI as a simple data collector, so that Windows users can still benchmark their UNIX servers. Results from the database can also be obtained and compared with local ones. Figure 2 shows the GUI as it edits a kernel while two remote-jobs are running. For full color screenshots please visit our homepage. The entire work flow with BenchIT is shown in figure 3.

2. Observed System Architectures

The environments for the discussed performance measurements are found in Opteron and Itanium 2 cluster. The system characteristics are shown in table 1. Parallelization in an Opteron cluster occurs at three different levels. The first one is hardware based, typically pipelining or
superscalarity. This level is handled internally by the processor and is therefore not discussed here. The next level is also processed internally. However, it is prescribed by the programmer or the compiler and uses an executable code. Examples are the Bit Level Parallelism and the usage of SIMD extensions. These extensions are not only supported by compiler libraries, but also used efficiently for code optimization. The highest level of parallelization is the utilization of several processors for solving the problem. For this purpose we studied optimized libraries with and without the usage of OpenMP. More information about the AMD64 family can be found as reported in the ZIH ([5]).

[1] `www.benchit.org`
[2] Integrated Development Environment

Table 1
Observed Systems

Architecture	IA-64	x86-64
Processor	Intel Itanium 2 Madison	AMD Opteron 248
Clockrate	1.4 GHz	2.2 GHz
RAM	4 GB	$\geq$ 2GB/node
Operating System	SuSE Enterprise Server 9	SuSE Enterprise Server 9
Kernel	2.6.5	2.6.5
available Compilers	GNU 4.0.1, Intel 9.0	GNU 4.0.1, Intel 9.0 Intel 8.1, Portland Group 6.0

Parallelization in the Itanium 2 Cluster starts on the same levels, but does not support SIMD-Extensions in IA-64 mode.

3. Performance Results

First, the maximum achievable performance for the measured algorithm, a matrix multiplication of dense matrices, on the introduced systems is checked. Therefore, the algorithm is measured with several optimized libraries. Then, the performance received from compiler generated code is compared against these results. The AMD Opteron has a peak performance of about 4 GFLOPS using ACML[3], which is close to its theoretical peak-performance. There is, however, no measured performance gain in the usage of packed-SSE2-instructions on AMD Opteron systems in a simple SSE2-implementation of the matrix multiplication. The Intel Itanium 2 processor reaches a peak performance of about 5.4 GFLOPS when using the Intel MKL[4]. All of the following results are gained with optimization-level -O3, unless otherwise noted.

3.1. Library Results

On AMD Opteron, ACML and ATLAS[5] show nearly the same performance approaching 4 GFLOPS whilst Intels MKL reaches a maximum of 1 GFLOPS. Also, the MKL breaks down abruptly at problem size 1290 and only recovers to a rate of 350 MFLOPS, as can be seen in figure 4. When running ATLAS on two processors, an acceleration of more than 1 is achieved for problem sizes larger then 200. Beyond that, it starts to speedup to 1.8. While the sequential work with ACML shows a continuous function for the different problem sizes, the parallel work shows a sawtooth-like behavior. On Intel Itanium 2, ATLAS reaches about 5 GFLOPS for large matrices. This value is reached for the first time durable on problem size 1600. Intels MKL shows a more continuous behavior. It reaches its maximum performance of 5.5 GFLOPS at problem size 300.

3.2. FORTRAN Results

FORTRAN is and has been optimized in numerical calculations. From the beginning it has supported programmers with data types and operations, which are still missing in other programming languages. Especially for scientific and numerical routines, FORTRAN is indispensable.
When comparing different commercial compilers on AMD Opteron, the peak performances between

[3] AMD Core Math Library
[4] Math Kernel Library
[5] Automatically Tuned Linear Algebra Software

them are nearly the same level. Both Intel 9.0 and Portland 6.0 show about 1.2 GFLOPS, which can be seen in figure 6. With a problem size between 100 and 300, Intel shows a better behavior, with different tableaus. The best performing permutation reaches a level of 1.1 GFLOPS. Pgfort also produces tableaus, but only for non-performing permutations. The GNU-Compiler only reaches a maximum of 970 MFLOPS out of the L1 Cache. When the data structures do not fit in any caches, the performance of all 3 compilers are between 20 and 420 MFLOPS.
Interestingly, the Intel Compiler 9.0 reaches a maximum of 1.2 GFLOPS, whereas its predecessor, icc 8.1, achieves 1.5 GFLOPS when the data fits in the L1 Cache. So a speedup of 0.8 can be seen in this generation step. Larger matrices that do not fit in the L1 Cache show only small performance differences, as seen in figure 7. Interesting behaviour is also seen in the Portland Compiler. While the most performing permutation under GNU and Intel Compilers is jki, the ikj-permutation exceeds the others when the datastructures fits in the Cache. When the data structures are larger, the results of the different compilers converge. This is also the reason why the best permutations do not show a tableau for problem sizes which run within the L2 Cache.
Running the same routine on an Intel Itanium 2, the Intel Compiler recognizes the matrix multiplication and replaces it with an optimized implementation. This allows a maximum performance of 5.1 GFLOPS and a sustained performance of 4.5 GFLOPS. While the Intel Compiler shows great results, the GNU compiler does not even reach 100 MFLOPS.

3.3. C Results

C is known for efficiency. Although it is also used for writing applications, it is the most popular programming language for writing system software. When compiling the matrix multiplication on the AMD Opteron with different compilers and a constant flag `-O3`, the Intel C-Compiler dominates the others with a maximal performance of 1.2 GFLOPS. The performance of the different permutations is as expected: ikj is the fastest (kji the slowest) with 1200(670) MFLOPS, when the data fits in the L1-Cache, and 420(25) MFLOPS, when the matrices are too large to fit any Caches. It is repeatedly shown that the predecessor offers a better performance for small matrices which fit in the L1-Cache. The Intel Compiler 8.1 reaches 1.3 GFLOPS.
The executable created by icc is a bit faster than the one generated by pgcc, which reaches a maximum performance of 1.15 GFLOPS. According to the FORTRAN results, when compiling with pgcc, the permutation kji runs the fastest, as long as the data fits into the L1-cache. As expected, however, when the size of the problem increases the permutation efficiency drops.
The GNU Compiler reaches a maximum performance of 960 MFLOPS. This value is achieved by 3 permutations and is relatively similar to the pgcc results for ikj and kij. When changing the the iccs compiler flags from `-O3` to `-O2` or `-Os`[6], the performance level stays at same. Using the gcc-option for 32-bit-code (`-m32`) results in a maximum performance of 550 MFLOPS. But by forcing the cpu to an AMD K8 (march=k8), the performance for large problem sizes remains above the level from using only `-O3`.
With Intel Itanium 2, the GNU Compiler does not reach a satisfactory result, with a performance peak of 190 MFLOPS. This peak, however, is twice as good as the result for the GNU FORTRAN Compiler. The Intel Compiler reaches, as expected, a higher level with a maximum of 650 MFLOPS, which is more then 3 times the gcc performance. When guaranteeing no overlapping data structures with flag `-fno-alias`, the performance increases rapidly with icc. The code is optimized: a maximum performance of 2.2 GFLOPS is reached. Sustained, it is still around 1.2 GFLOPS for all permutations except kij, which shows its normal behavior. Overall, problem sizes which are

[6] enable speed optimizations, but disable some optimizations which increases code size for small speed benefit

multiples of 16 show the best performance for the optimized permutations, as seen in figure 5.

4. Conclusion and further work

BenchIT is a powerful and flexible tool which allows performance measurements of POSIX.1 conform computers. It supports users to write performance measuring kernels as well as compile and run kernels in different environments. Using BenchIT, one can easily plot results to share them among users worldwide. This benchmarking suite, also userfriendly and flexible to use, allows easy system characteristic comparisons, such as the type of RAM, the size of caches, or the processors clock rate.
Clear conventions, extendable configurations, and its independence from specific platforms allow this project to advance into many directions. Further development in this area will offer new possibilities to measure and compare different systems and environments. New batch-environments will be specified, new measurement kernels developed and, of course, the supportive tools will advance even further.

References

[1] JUCKELAND, Guido; BÖRNER, Stefan; KLUGE, Michael; KÖLLING, Sebastian; NAGEL, Wolfgang E.; PFLÜGER, Stefan; RÖDING, Heike; SEIDL, Stefan; WILLIAM, Thomas; WLOCH, Robert:
BenchIT - Performance Measurements and Comparison for Scientific Applications.
In: JOUBERT, Gerhard R. (Hrsg.) ; NAGEL, Wolfgang E. (Hrsg.) ; PETERS, F. J. (Hrsg.) ; WALTER, W. V. (Hrsg.): *PARCO* Bd. 13, Elsevier, 2003. –
ISBN 0–444–51689–1, S. 501–508

[2] JUCKELAND, Guido; KLUGE, Michael; NAGEL, Wolfgang E.; PFLÜGER, Stefan:
Performance Analysis with BenchIT: Portable, Flexible, Easy to Use.
In: *QEST*, IEEE Computer Society, 2004. –
ISBN 0–7695–2185–1, S. 320–321

[3] INNOVATIVE COMPUTING LABORATORY, UNIVERSITY OF TENNESSEE.
PAPI Website.
`http://icl.cs.utk.edu/papi`

[4] ZIEGLER, Heinz; MOHR, Bernd
CENTRAL INSTITUTE FOR APPLIED MATHEMATICS (ZAM) AT THE RESEARCH CENTER JUELICH, GERMANY.
PCL Website.
`http://www.fz-juelich.de/zam/PCL/`

[5] JUCKELAND, Guido
CENTER FOR INFORMATIONSERVICES AND HIGHPERFORMANCECOMPUTING (ZIH) AT THE TECHNISCHE UNIVERSITÄT DRESDEN, GERMANY.
The AMD64 Chip Family: Concepts and First Measurements .
ZHR-R-0403

Figure 1. Autodetection of different environment settings

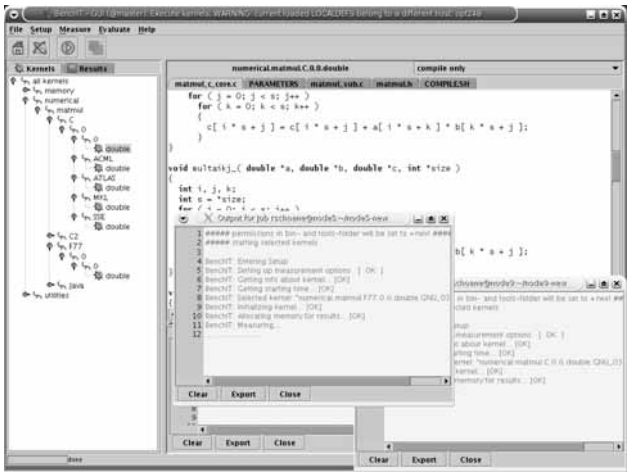

Figure 2. GUI

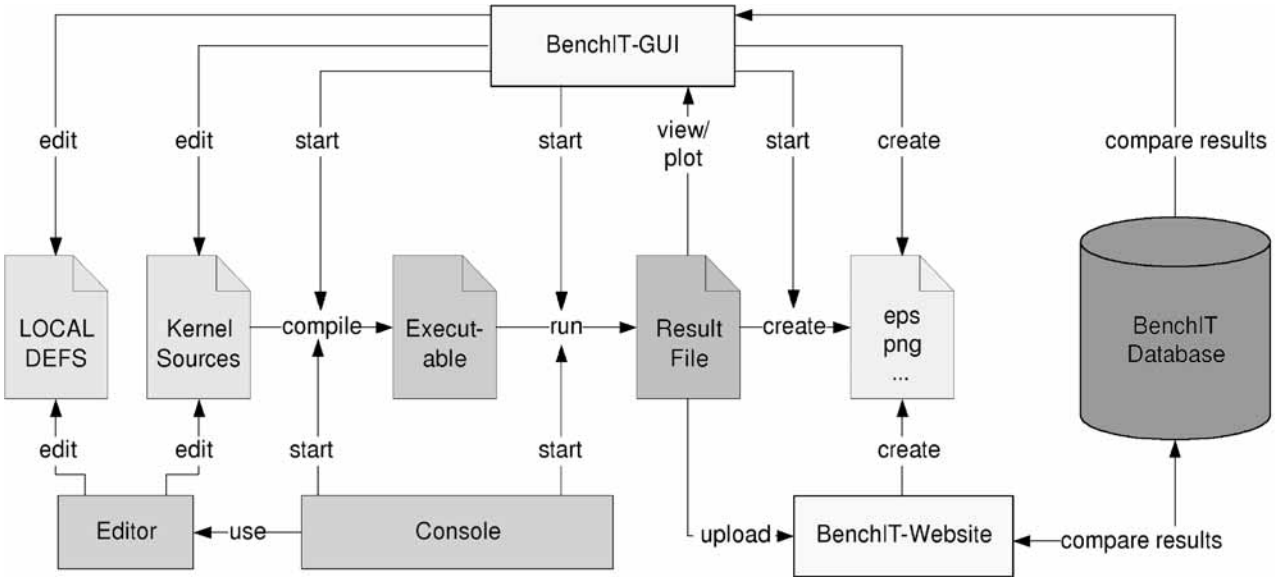

Figure 3. BenchIT - functional overview

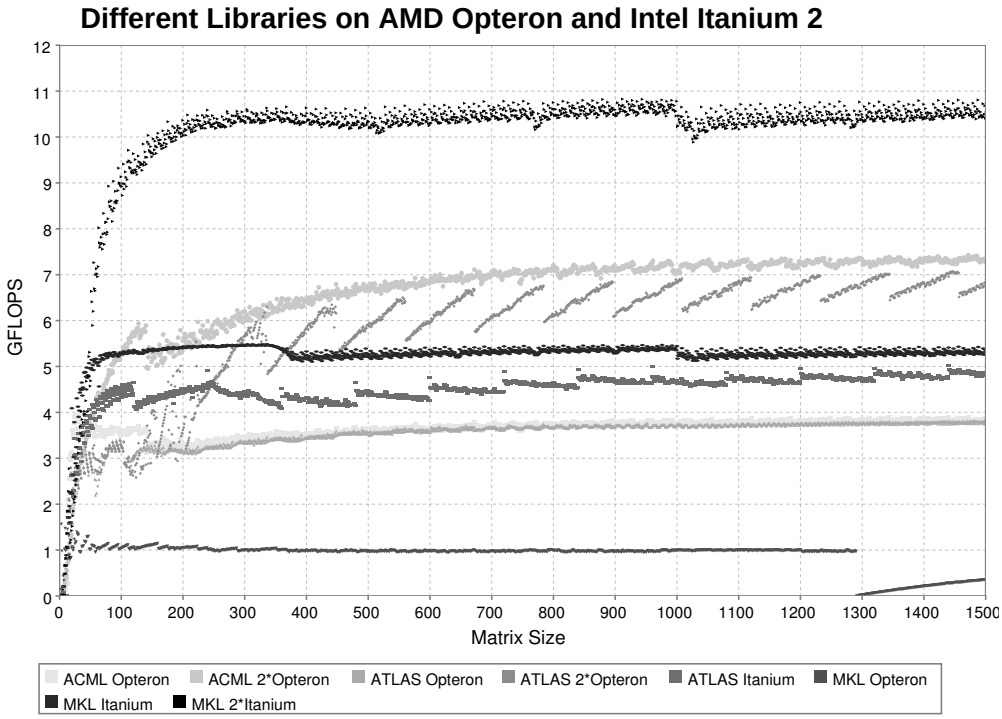

Figure 4. DGEMM with MKL, ATLAS and ACML on AMD Opteron and Intel Itanium 2

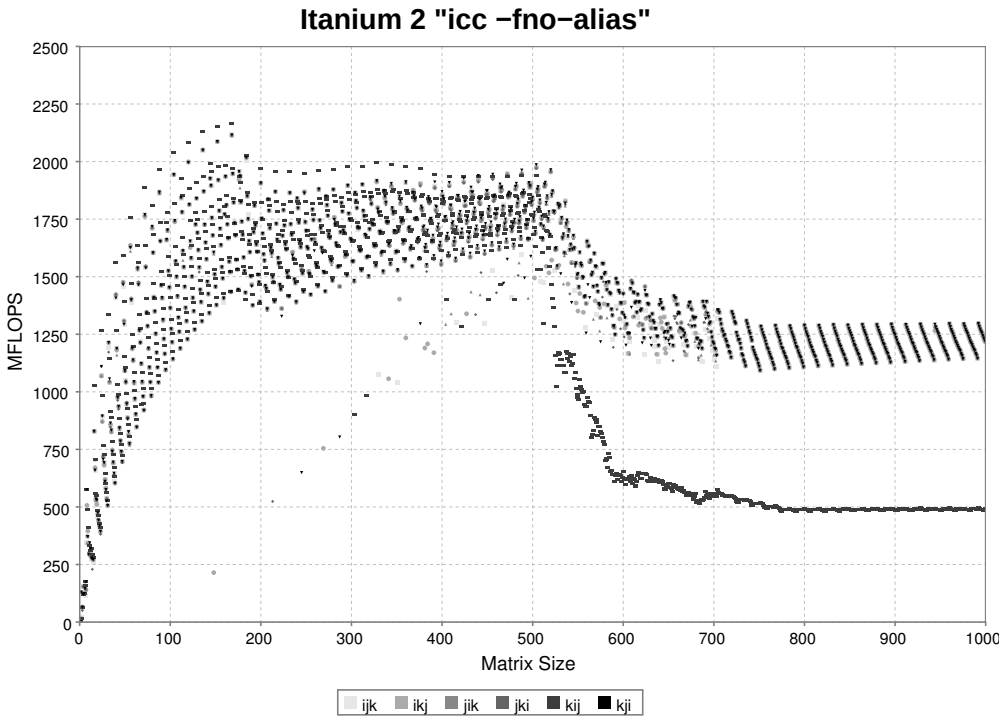

Figure 5. Matrix multiplication with ”icc -fno-alias” on Intel Itanium 2

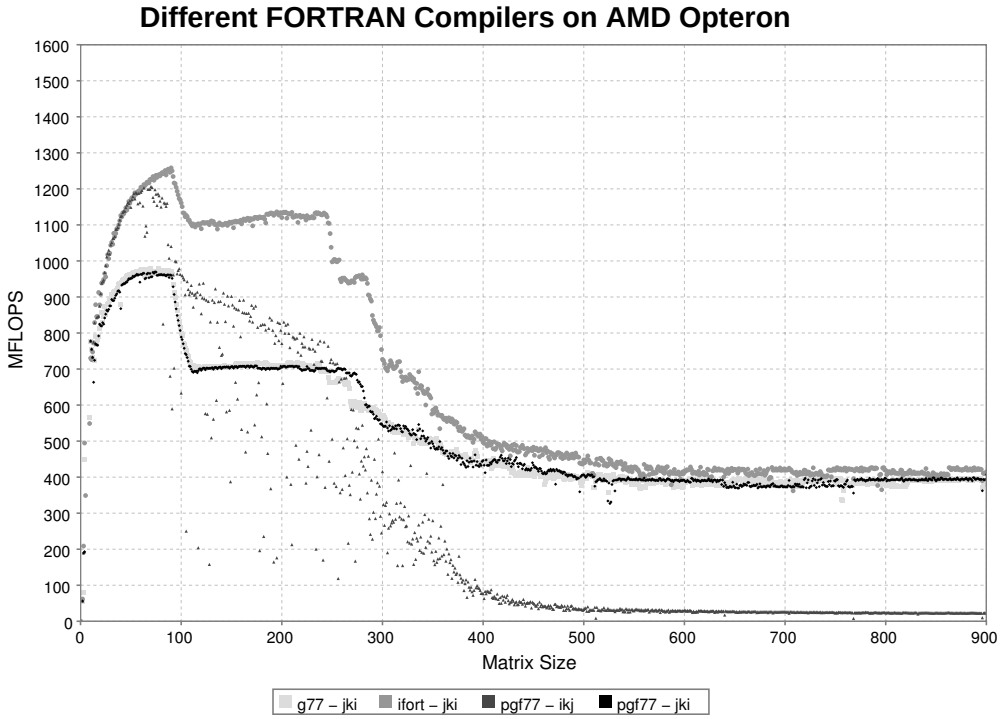

Figure 6. Matrix multiplication with different compilers on AMD Opteron

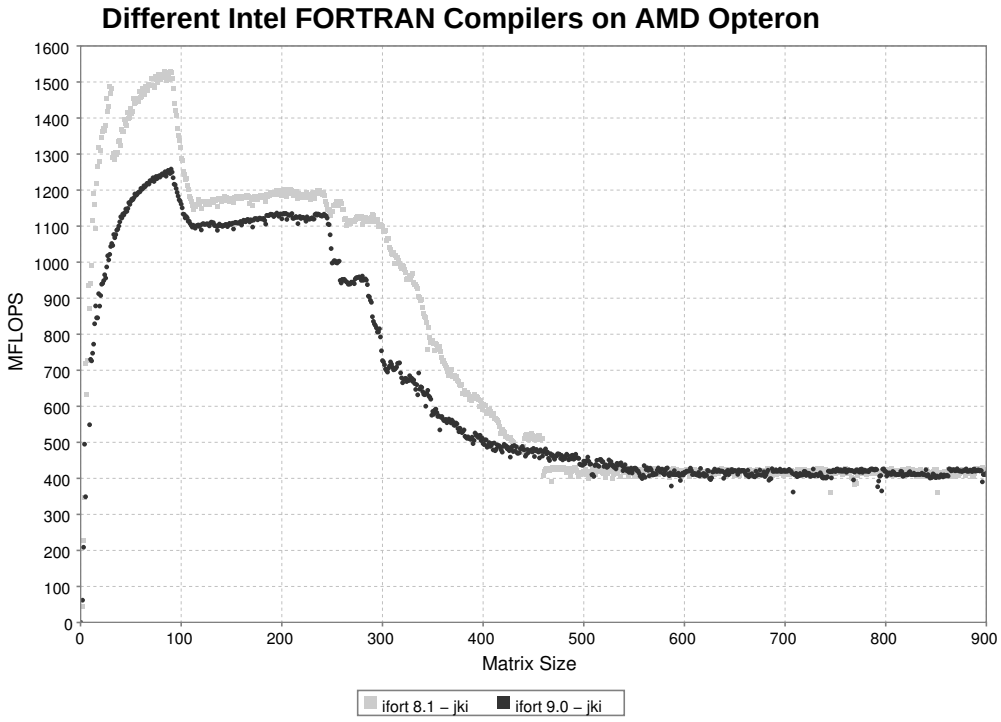

Figure 7. Matrix multiplication on AMD Opteron with different versions of ifort